

A Universal Parallel Two-Pass MDL Context Tree Compression Algorithm

Nikhil Krishnan, *Student Member, IEEE*, and Dror Baron, *Senior Member, IEEE*

Abstract—Computing problems that handle large amounts of data necessitate the use of lossless data compression for efficient storage and transmission. We present a novel lossless universal data compression algorithm that uses parallel computational units to increase the throughput. The length- N input sequence is partitioned into B blocks. Processing each block independently of the other blocks can accelerate the computation by a factor of B , but degrades the compression quality. Instead, our approach is to first estimate the minimum description length (MDL) context tree source underlying the entire input, and then encode each of the B blocks in parallel based on the MDL source. With this two-pass approach, the compression loss incurred by using more parallel units is insignificant. Our algorithm is work-efficient, i.e., its computational complexity is $O(N/B)$. Its redundancy is approximately $B \log(N/B)$ bits above Rissanen’s lower bound on universal compression performance, with respect to any context tree source whose maximal depth is at most $\log(N/B)$. We improve the compression by using different quantizers for states of the context tree based on the number of symbols corresponding to those states. Numerical results from a prototype implementation suggest that our algorithm offers a better trade-off between compression and throughput than competing universal data compression algorithms.

Index Terms—big data, computational complexity, data compression, distributed computing, minimum description length, parallel algorithms, redundancy, two-pass code, universal compression, work-efficient algorithms.

I. INTRODUCTION

A. Motivation

The emergence of distributed cloud computing and big data problems raises new challenges in data storage and communication. In such distributed computing settings, the data may be processed remotely in clusters and the

results are streamed to the end user through a network. The use of data streaming in big data problems makes it imperative to use fast lossless data compression algorithms whose primary features include good compression quality and high throughput.

Some applications of fast compression include internet backbone data compression and compression in high volume data generation applications such as scientific computing. Data can be compressed rapidly near the source of data generation, and can be transmitted through band limited channels. This compression scheme can reduce energy consumption and bandwidth requirements of the network.

Several techniques are available to improve the throughput, such as hardware acceleration [3], algorithmic approximations, and computer architecture optimizations [4–7]. Although these acceleration, approximation, and optimization techniques may accelerate compression, there are many systems where these do not suffice either due to limited speed up or poor compression quality. Ultimately, in order for lossless compression to become appealing for a broader range of applications, we must concentrate more on efficient new algorithms.

Over the last decade, inexpensive multi core processors such as *graphics processing units* (GPUs) have become available, and parallelization is a possible direction for fast compression algorithms. By compressing in parallel, we may obtain algorithms that are faster by orders of magnitude. However, with a naive parallel algorithm, which consists of partitioning the original input into B blocks and processing each block independently of the other blocks, increasing B degrades the compression quality [8]. Therefore, naive parallel compression has limited potential. Sharing information across blocks can improve the compression quality of data [9].

B. Related work

Stassen and Tjalkens [10] proposed a parallel compression algorithm based on context tree weighting [11] (CTW), where a common finite state machine (FSM)

N. Krishnan and D. Baron are with the Department of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC, 27695 USA. E-mail: {nkrishn, barondror}@ncsu.edu.

This work was supported in part by the National Science Foundation under Grant CCF-1217749 and in part by the U.S. Army Research Office under Grants W911NF-04-D-0003 and W911NF-14-1-0314. Portions of this paper were presented at the IEEE International Symposium on Information Theory, Honolulu, Hawaii, June 2014 [1], and at the IEEE Global Conference on Signal and Information Processing, Atlanta, Georgia, December 2014 [2].

determines for each symbol which processor should process it. Since the FSM processes the original length- N input in $O(N)$ time, Stassen and Tjalkens' method does not support scalable data rates.

Franaszek et al. [8] proposed a parallel compression algorithm, which is related to LZ77 [12], where the construction of a dictionary is divided between multiple processors. Unfortunately, the redundancy (excess coding length above the entropy rate) of LZ77 is high.

Finally, Willems [13] proposed a variant of CTW with $O(ND/B)$ time complexity, where D is the maximal context depth that is processed. Unfortunately, Willems' approach will not compress as well as CTW, because probability estimates will be based on partial information in between synchronizations of the context trees.

C. Contributions

This paper presents a novel minimum description length [14] (MDL) data compression algorithm that coordinates multiple computational units running in parallel, such that the compression loss incurred by using more computational units is insignificant. Our main contributions are (i) our algorithm is *work-efficient* [15], i.e., it compresses B length- (N/B) blocks in parallel with $O(N/B)$ time complexity; (ii) the redundancy of our algorithm is approximately $B \log(N/B)$ bits above the lower bounds on the best achievable redundancy; (iii) we improve the compression quality by using different quantizers for states of the context tree based on the number of symbols corresponding to those states; (iv) using a serial (non-parallel) prototype implementation, we compare the compression and throughput as a function of the number of parallel computational units available; and (v) our algorithm has the useful property of random access [16], where any part of the compressed file can be decompressed without decompressing the entire file. Numerical results show that our *parallel two-pass MDL* (PTP-MDL) algorithm provides a better trade-off between compression and throughput, which makes this algorithm attractive for big data problems.

The remainder of the paper is organized as follows. We review preliminary material in Section II, describe our PTP-MDL algorithm [1, 17] in Section III, discuss numerical results in Section IV, and conclude in Section V.

II. DATA COMPRESSION PRELIMINARIES

A. Universal data compression

Lower bounds on the redundancy serve as benchmarks for compression quality. Consider length- N sequences

x^N generated by a stationary ergodic source over a finite alphabet \mathcal{X} , i.e., $x^N \in \mathcal{X}^N$. For an individual sequence x^N , the *pointwise redundancy* with respect to (w.r.t.) a class \mathcal{C} of source models is

$$\rho(x^N) \triangleq l(x^N) - N\widehat{H}_x,$$

where $l(x^N)$ is the length of a uniquely decodable code [12] for x^N , and \widehat{H}_x is the entropy rate of x^N w.r.t. the best model in \mathcal{C} with parameters set to their *maximum likelihood* (ML) estimates. Weinberger et al. [18], whose result was improved upon later, proved for a source with K (unknown) parameters that

$$\rho(x^N) \geq \frac{K}{2}(1 - \epsilon) \log(N), \quad (1)$$

for any $\epsilon > 0$, except for a set of inputs whose probability vanishes as $N \rightarrow \infty$, where $\log(\cdot)$ denotes the base-2 logarithm. Similarly, Rissanen [19] proved that, for universal compression of independent and identically distributed (i.i.d.) sequences, the worst case redundancy (WCR) is

$$\rho(x^N) \geq \frac{|\mathcal{X}| - 1}{2} \log(N) + C_{|\mathcal{X}|} + o(1), \quad (2)$$

where $|\mathcal{X}|$ denotes cardinality of \mathcal{X} , the constant $C_{|\mathcal{X}|}$ was specified, and the $o(1)$ term vanishes as N grows. Rissanen's result for the WCR holds for any sequence whose ML estimates satisfy the central limit theorem, where we can replace $|\mathcal{X}| - 1$ in the expression for the WCR (2) of i.i.d. sequences with K . Because i.i.d. models are too simplistic for modeling real-world data, we use tree sources instead.

B. Tree sources

Let x_i^j denote the *sequence* x_i, x_{i+1}, \dots, x_j where $x_k \in \mathcal{X}$ for $i \leq k \leq j$. Let \mathcal{X}^* denote the set of finite-length sequences over \mathcal{X} . Define a *context tree source* $\{\mathcal{S}, \Theta\}$ [11] as a finite set of sequences called states $\mathcal{S} \subset \mathcal{X}^*$ that is complete and proper [11, p.654], and a set of conditional probabilities $\Theta = \{p(\alpha|s) : \alpha \in \mathcal{X}, s \in \mathcal{S}\}$. We say that s *generates* symbols following it. Because \mathcal{S} is complete and proper, the sequences of \mathcal{S} can be arranged as leaves on an $|\mathcal{X}|$ -ary tree [15] (Figure 1); the unique state s that generated x_i can be determined by entering the tree at the root, first choosing branch x_{i-1} , then branch x_{i-2} , and so on, until some leaf s is encountered. Let $D \triangleq \max_{s \in \mathcal{S}} |s|$ be the *maximum context depth*. Then the string x_{i-D}^{i-1} uniquely determines the current state s ; the previous symbols x_{i-L}^{i-1} ($L \leq D$) that uniquely determine the current state s are called the *context*, and L is called the *context depth* for state s .

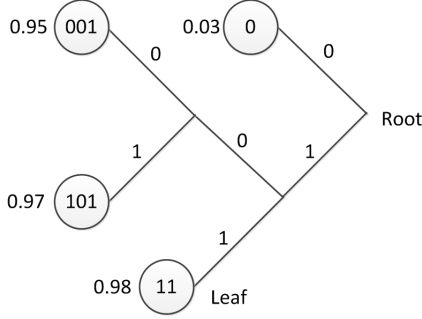


Figure 1: A tree source over $\mathcal{X} = \{0, 1\}$. The states are $\mathcal{S} = \{0, 11, 001, 101\}$ and the conditional probabilities are $p(x_i = 1|0) = 0.03$, $p(x_i = 1|11) = 0.98$, $p(x_i = 1|001) = 0.95$, and $p(x_i = 1|101) = 0.97$. Data generated by this tree are quite compressible, because conditional probabilities are close to 0 and 1.

C. Semi-predictive and two-pass data compression

Semi-predictive compression: Consider a tree source structure \mathcal{S} whose explicit description requires $l_{\mathcal{S}}$ bits, and denote the probability of the input sequence x^N conditioned on the tree source structure \mathcal{S} by $p_{\mathcal{S}}(x^N)$. Using \mathcal{S} , the coding length required for x^N is $l_{\mathcal{S}} + l_{\mathcal{C}}$, where $l_{\mathcal{C}} = -\log(p_{\mathcal{S}}(x^N))$ is the coding length. Let the MDL tree source structure $\hat{\mathcal{S}}$ be the tree structure that provides the shortest description of the data, i.e.,

$$\hat{\mathcal{S}} \triangleq \arg \min_{\mathcal{S} \in \mathcal{C}} \{l_{\mathcal{S}} + l_{\mathcal{C}}\},$$

where \mathcal{C} is the class of tree source models being considered. The semi-predictive approach [20–22] processes the input x^N in two passes. Pass I first estimates $\hat{\mathcal{S}}$ by context tree pruning, which minimizes the coding length. The structure of $\hat{\mathcal{S}}$ is then encoded explicitly. Pass II uses $\hat{\mathcal{S}}$ to encode the sequence x^N sequentially, where the parameters $\hat{\Theta}$ are estimated while encoding x^N . The decoder first determines $\hat{\mathcal{S}}$, and afterwards uses it to decode x^N sequentially from the two-pass code.

Two-pass compression: In contrast to the semi-predictive approach, which only encodes the structure of $\hat{\mathcal{S}}$ in Pass I, our two-pass approach also encodes the parameter values $\hat{\Theta}$. We use a two-pass approach instead of a semi-predictive approach, because estimating B sets of parameters in parallel, one for encoding each of the B blocks in Pass II, has $\rho(x^N) \approx 0.5B|\mathcal{S}|\log(N/B)$, whereas the two-pass approach has $\rho(x^N) \approx 0.5|\mathcal{S}|\log(N)$, and the latter redundancy is smaller. Despite the parallel nature of our algorithm, it incurs a single redundancy term for lack of knowledge of the parameters in Pass I instead of B redundancy terms

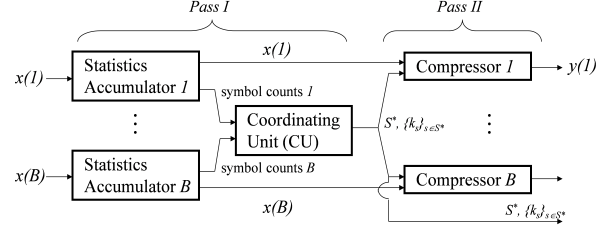


Figure 2: Block diagram of the PTP-MDL encoder.

in Pass II.

III. PTP-MDL ALGORITHM

In order to keep the presentation simple, we restrict our attention to a binary alphabet, i.e., $\mathcal{X} = \{0, 1\}$; the generalization to non-binary alphabets is straightforward. We will show that PTP-MDL has $O(N/B)$ time complexity when we restrict $D \leq \log(N/B)$, while still approaching the pointwise redundancy bound (1). These properties enable scalable data rates without a factor- B increase in the redundancy.

A. Overview

Encoder: A block diagram of the PTP-MDL encoder is shown in Figure 2. In Pass I, the PTP-MDL encoder employs B computational units called *parallel units* (PUs) that work in parallel to accumulate statistical information on B blocks in $O(N/B)$ time, and a *coordinating unit* (CU) that controls the PUs and computes the MDL source estimate $\{\hat{\mathcal{S}}, \hat{\Theta}\}$.

Without loss of generality, we assume $N/B \in \mathbb{Z}^+$. Define the B blocks as $x(1) = x_1^{N/B}, x(2) = x_{N/B+1}^{2N/B}, \dots, x(B) = x_{(B-1)N/B+1}^N$. Parallel unit b , where $b \in \{1, \dots, B\}$, first computes for each depth- D context s the *block symbol counts* $n_s^\alpha(b)$, which are the number of times α is generated by s in $x(b)$,

$$n_s^\alpha(b) \triangleq \sum_{i=(b-1)(N/B)+D+1}^{b(N/B)} 1_{\{x_{i-D}^{i-1} = s\alpha\}}, \alpha \in \mathcal{X}, \quad (3)$$

where $s\alpha$ denotes concatenation of s and α , and $1_{\{\cdot\}}$ is the indicator function. For each state s such that $|s| < D$, the CU either retains the children states $0s$ and $1s$ in the MDL source, or prunes them and only retains s , whichever results in a shorter coding length. Details of the pruning decision appear in Section III-C. Note that a single encoder compressing all N symbols has access to the last D symbols from the previous block as context for encoding the first D symbols of the current block (except for the first block). However, the MDL source of a single

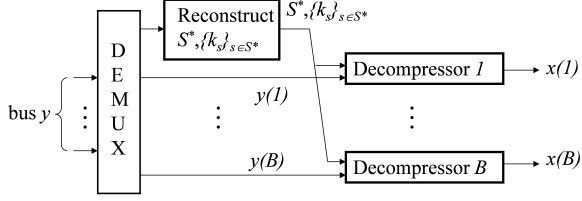


Figure 3: Block diagram of the PTP-MDL decoder.

encoder is suboptimal in PTP-MDL, because this source does not reflect the actual symbols compressed by PTP-MDL.

In Pass II, each of the B blocks is compressed by a PU. For each symbol $x_i(b)$, PU b first determines the generator state $G_i(b)$, the state s that generated the symbol $x_i(b)$. PU b then assigns $x_i(b)$ a probability according to the parameters that were estimated by the CU in Pass I, and sequentially feeds the probability assignments to an arithmetic encoder [12].

Decoder: A block diagram of the PTP-MDL decoder is shown in Figure 3. The structure of the decoder is similar to that of Pass II. The approximated MDL source structure \hat{S} and quantized parameters $\hat{\Theta}$ are first derived from the parallel source description (see Section III-B). Then, the B blocks are decompressed by B decoding blocks. In decoding block b , each symbol $x_i(b)$ is sequentially decoded by determining the generator state $G_i(b)$, assigning a probability to $x_i(b)$ based on the parameter estimates, and applying an arithmetic decoder [12].

B. Parallel source description

Two-pass codes in the PTP-MDL algorithm: Having received the block symbol counts $n_s^\alpha(b)$ from the PUs (3), the CU computes the *symbol counts* generated by state s in the entire sequence x^N ,

$$n_s^\alpha = \sum_{b=1}^B n_s^\alpha(b), \quad \alpha \in \mathcal{X}. \quad (4)$$

The CU can then compute the ML parameter estimates of $p(1|s)$ and $p(0|s)$,

$$\theta_s \triangleq \theta_s^1 = \frac{n_s^1}{n_s^0 + n_s^1} \quad \text{and} \quad \theta_s^0 = 1 - \theta_s^1,$$

respectively. The ML parameter estimates for each state

s are quantized into one of

$$\begin{aligned} K_s &\triangleq \left\lceil \sqrt{2\pi^2 \ln(2) \left(\frac{1}{2} - \frac{3}{16 \ln(2)} \right) N} \right\rceil \\ &\approx \lceil 1.772\sqrt{N} \rceil \end{aligned} \quad (5)$$

representation levels based on Jeffreys' prior such that each bin has the same probability mass [23], where $\lceil \cdot \rceil$ denotes rounding up. Jeffreys' prior for the scalar parameter θ is $p(\theta) \propto 1/\sqrt{\theta(1-\theta)}$, which is the arcsine distribution, and is almost flat for interior values of the parameter, $\theta \in (0, 1)$. The representation levels and bin edges are computed using a closed form quantizer [23]. The *bin index* and representation level for state s are denoted by k_s and r_s , respectively. Denoting the quantized ML estimate of θ_s^α by $\hat{\theta}_s^\alpha$, we have $\hat{\theta}_s^1 = r_s$ and $\hat{\theta}_s^0 = 1 - r_s$. Recall that, at the end of Pass I, the CU has computed the MDL structure estimate \hat{S} . If $s \in \hat{S}$, then the first pass of the two-pass code for symbols generated by s consists of encoding k_s with $\log(K_s)$ bits. The WCR using this quantization approach is 1.047 bits per state above Rissanen's redundancy bound [17, 19, 23].

In Pass II, which implements the second pass of the two-pass code, each PU b encodes its block $x(b)$ sequentially. For each symbol $x_i(b)$, PU b determines the generator state $G_i(b)$. The symbol $x_i(b)$ is encoded according to the probability assignment $\hat{p}(x_i(b)) \triangleq \hat{\theta}_{G_i(b)}^{x_i(b)}$ with an arithmetic encoder [12]. Thus, the probability assigned by all B PUs to the symbols in x^N whose generator state is s is

$$\prod_{b=1}^B \prod_{\{i: G_i(b)=s, i>D\}} \hat{p}(x_i(b)) = (r_s)^{n_s^1} (1 - r_s)^{n_s^0}. \quad (6)$$

Equation (6) provides the same coding length for two-pass codes in a parallel compression system as we would obtain in a serial system [17].

Coding lengths in Passes I and II: In Pass I, the structure \hat{S} is described with the *natural code* [11]. For a binary alphabet, $|\text{natural}_{\hat{S}}| \leq 2|\hat{S}| - 1$ bits; this is the *model redundancy* of PTP-MDL. The parameters $\hat{\Theta}$ are described as the $|\hat{S}|$ indices k_s in the order in which the leaves of \hat{S} are reached in a depth-first search [15]; this description can be implemented with arithmetic coding [12]. The corresponding coding length is the *parameter redundancy* of PTP-MDL. We denote the length of the descriptions of \hat{S} and $\hat{\Theta}$ generated in Pass I by l_S^I bits. Using (5),

$$\begin{aligned} l_S^I &= |\text{natural}_{\hat{S}}| + |\hat{S}| \log(K_s) \\ &\lesssim \lceil 2|\hat{S}| - 1 \rceil + |\hat{S}| \left[\log(1.772) + \frac{1}{2} \log(N) \right], \end{aligned} \quad (7)$$

where \lesssim denotes less than or approximately equal to.

In Pass II, the coding length is mainly determined by symbol probabilities conditioned on generator states as given by (6). There are two additional terms that affect the coding length in Pass II. First, *coding redundancy* for each arithmetic encoder with $\log(N)$ bits of precision requires $O(1) \leq 2$ bits [12]. Second, *symbols with unknown context* at the beginning of $x(b)$; we encode the first D symbols of each block $x(b)$ directly using D bits per block. Denoting the combined length of all B codes in Pass II by l_C^{II} bits, we have

$$l_C^{II} \lesssim B \cdot (D + 2) - \sum_{s \in S} [n_s^1 \log(r_s) + n_s^0 \log(1 - r_s)]. \quad (8)$$

Combining (7) and (8), we have the following result for the redundancy.

Theorem 1. (Theorem 18 in Baron [17].) *The pointwise redundancy of the PTP-MDL algorithm over the ML entropy of the input sequence x^N w.r.t. the MDL source structure \hat{S} , which is an element in the class of tree sources of depth $D \leq \log(N/B)$, satisfies*

$$\rho_P(x^N) \leq B \left[\log \left(\frac{N}{B} \right) + 2 \right] + \frac{|\hat{S}|}{2} [\log(N) + O(1)],$$

where the subscript P denotes the PTP-MDL compression algorithm.

Proof outline. (For a detailed presentation, see pages 107 – 115 in Baron [17].) The components of redundancy w.r.t. \hat{S} are as follows.

- The model redundancy of the natural code is at most $2|\hat{S}| - 1$ bits.
- From Section II-A, we saw that Rissanen's bound for the WCR (2) is $\frac{1}{2} \log(N) + O(1)$ bits per state. The parameter redundancy introduced by two-pass codes based on Jeffreys' prior described in Section III-B is within 1.047 bits per state above Rissanen's redundancy bound [23].
- Each PU b encodes the first $D = \log \left(\frac{N}{B} \right) + O(1)$ bits directly.
- Because the arithmetic computations are performed with finite precision, the outcome \hat{S} may not be an MDL tree source structure for x^N . Theorem 15 in Baron [17] indicates that the coding length with \hat{S} is at most $O(1)$ bits more than the coding length obtained from the MDL context tree source structure for x^N .
- The upper bound of arithmetic coding redundancy is 2 bits per PU [12].

In combination, these steps yield the WCR result of Theorem 1. ■

The pointwise redundancy of naive parallel compression, which is denoted by $\rho_N(x^N)$ and is defined in Section I-A, is worse compared to $\rho_P(x^N)$.

Proposition 1. *The pointwise redundancy of the naive parallel compression algorithm over the ML entropy of the input sequence x^N w.r.t. the MDL source structure \hat{S} , which is an element in the class of tree sources of depth $D \leq \log(N/B)$, satisfies*

$$\begin{aligned} \rho_N(x^N) &\leq B \left[\log \left(\frac{N}{B} \right) + 2 \right] \\ &+ B \left[\frac{|\hat{S}_n|}{2} \left[\log \left(\frac{N}{B} \right) + O(1) \right] \right], \end{aligned}$$

where \hat{S}_n is the estimated tree structure with the largest number of states among the B tree structures.

Proof outline. The redundancy analysis is similar to Theorem 1, except that the model redundancy and parameter redundancy in the naive parallel compression algorithm are B times the respective redundancies in the PTP-MDL algorithm as we estimate separate models for data of length (N/B) in each of the B PUs. ■

C. Pass I

Computing block symbol counts: Computational unit b computes $n_s^\alpha(b)$ for all 2^D depth- D leaf contexts s . In order for PU b to compute all block symbol counts in $O(N/B)$ time, we define the *context index* $c_i(b)$ of the symbol $x_i(b)$ as

$$c_i(b) \triangleq \sum_{j=0}^{D-1} 2^j x_{j+i-D}(b), \quad (9)$$

where $i \in \{D+1, \dots, N/B\}$ and $x_{j+i-D}(b) \in \{0, 1\}$, hence $c_i(b) \in \{0, \dots, 2^D - 1\}$. Note that $c_i(b)$ is the binary number represented by the context $s = x_{i-D}^{i-1}(b)$. Hence, it can be used as a pointer to the address containing the block symbol count $n_s^\alpha(b)$ for $s = x_{i-D}^{i-1}(b)$. Moreover, the property

$$c_{i+1}(b) = \frac{c_i(b)}{2} + 2^{D-1} x_i(b) - \frac{x_{i-D}(b)}{2} \quad (10)$$

enables the computation of all $N/B - D$ context indices of the symbols of $x(b)$ in $O(N/B)$ time complexity.

Constructing context trees: Because we restrict our attention to depth- D contexts, it suffices for PU b to compute $\{n_s^\alpha(b)\}_{\alpha \in \mathcal{X}, s \in \mathcal{X}^D}$, all the block symbol counts of all the leaf contexts of a full depth- D context tree. Information on internal nodes of the context tree, whose

depth is less than D , is computed from the block symbol counts of the leaf contexts.

If $|s| = D$, then the CU gets $\{n_s^\alpha(b)\}_{\alpha \in \mathcal{X}}$ from the PUs and computes n_s^α with (4). Alternatively, $|s| < D$, and the CU recursively derives n_s^α by adding up the symbol counts of children states, i.e.,

$$n_s^\alpha = n_{0s}^\alpha + n_{1s}^\alpha, \quad \forall \alpha \in \mathcal{X}. \quad (11)$$

Computing the MDL source $\{\hat{\mathcal{S}}, \hat{\Theta}\}$: For each state s , we either retain the children states $0s$ and $1s$ in the tree or merge them into a single state, according to which decision minimizes the coding length. The coding length l_s of the two-pass code that describes the symbols generated by s is

$$l_s = \overbrace{\log(K_s)}^{\text{Pass I}} - \overbrace{n_s^0 \log(1 - r_s) - n_s^1 \log(r_s)}^{\text{Pass II}}. \quad (12)$$

We now derive the coding length required for state s , which is denoted by MDL_s . For $|s| = D$, n_s^0 and n_s^1 are computed with (4), l_s is computed with (12), and $\text{MDL}_s = l_s$. For $|s| < D$, we compute n_s^α hierarchically with (11), after already having processed the children states. In order to decide whether to prune the tree, we compare $\text{MDL}_{0s} + \text{MDL}_{1s}$ with l_s . Because retaining an internal node requires the natural code [11] to describe that node (with 1 bit),

$$\text{MDL}_s = \begin{cases} l_s & \text{if } |s| = D \\ 1 + \min\{\text{MDL}_{0s} + \text{MDL}_{1s}, l_s\} & \text{else} \end{cases}.$$

In terms of the natural code, if $|s| = D$, then s is a leaf of the full depth- D context tree, and its natural code is empty; else $|s| < D$, and the natural code requires 1 bit to encode whether $s \in \mathcal{S}$. The symbols generated by s are encoded either by retaining the children states (this requires a coding length of $\text{MDL}_{0s} + \text{MDL}_{1s}$ bits), or by pruning the children states and retaining state s with coding length l_s . If $|s| = D$, then we do not process deeper contexts. The context tree pruning has $O(N/B)$ time complexity because the tree has $O(N/B)$ states.

D. Pass II

In Pass II, PU b knows $\hat{\mathcal{S}}$ and $\{r_s\}_{s \in \hat{\mathcal{S}}}$. PU b encodes $x(b)$ sequentially; for each symbol $x_i(b)$, it determines the generator state $G_i(b)$. An $O(N/B)$ algorithm for determining $G_i(b)$ for all the symbols of $x(b)$ utilizing (9,10) is described by Baron [17]. After determining $G_i(b)$, the symbol $x_i(b)$ is encoded according to the probability assignment $\hat{p}(x_i(b)) \triangleq \hat{\theta}_{G_i(b)}^{x_i(b)}$ with an arithmetic encoder [12]. In order to have $O(N/B)$ time complexity and $O(1)$ expected coding redundancy per PU, arithmetic coding is performed with $\log(N)$ bits

of precision [12], where we assume that the hardware architecture performs arithmetic with $\log(N)$ bits of precision in $O(1)$ time.

E. Decoder

The B decoding blocks can be implemented on B PUs. Decoding block b decodes $x(b)$ sequentially; for each symbol $x_i(b)$, it determines $G_i(b)$. The same $O(N/B)$ algorithm used in Pass II for determining $G_i(b)$ for all the symbols of $x(b)$ can be used in the B decoding blocks. After determining $G_i(b)$, the symbol $x_i(b)$ is decoded according to the probability assignment $\hat{p}(x_i(b)) \triangleq \hat{\theta}_{G_i(b)}^{x_i(b)}$ with an arithmetic decoder [12] that has $O(N/B)$ time complexity. We have the following result for the overall time complexity of the encoder and decoder.

Theorem 2. (Theorem 16 in Baron [17].) *With computations performed with $\log(N)$ bits of precision defined as $O(1)$ time, the PTP-MDL encoder and decoder each require $O(N/B)$ time.*

Proof outline. (For a detailed presentation, see pages 116 – 120 in Baron [17].) For the algorithm to run in $O(N/B)$ time, all subroutines in Pass I and Pass II should run in $O(N/B)$ time. An outline of the time complexity analysis for the subroutines is as follows.

- Computing the block symbol counts in (3) has $O(N/B)$ complexity as x^N is divided among B PUs, and the context index of each symbol in x^N can be updated in $O(1)$ time utilizing (9,10).
- Adding up the block symbol counts in (4) for each state can be performed in $O(1)$ time using a pipelined adder tree [17]. Since we limit the maximum number of states to be $2^D = (N/B)$, the computational complexity is $O(N/B)$.
- The context tree pruning processes $O(2^D) = O(N/B)$ contexts, and each function call decides to prune based on $O(1)$ computations involving equations (11, 12).
- A generator look up table, which maps the generator state $G_i(b)$ to the probability assignment $\hat{p}(x_i(b))$, is constructed in $O(N/B)$ time utilizing (9,10).
- Pass II has $O(N/B)$ time complexity when context indices and a generator look up table are used to assign probabilities to each of the B arithmetic encoders. The decoder also has $O(N/B)$ time complexity, because its structure is similar to that of Pass II.

In combination, these steps yield the $O(N/B)$ time complexity result of Theorem 2. ■

F. Quantization schemes for better compression

In equation (5) for estimating K_s , we used a conservative estimate for each *context population*, which is defined as the number of symbols for a given context, i.e., $n_s = n_s^0 + n_s^1$, where we implicitly assumed $n_s = N$. However, the number of symbols for a particular context is often much smaller than N . We can improve the compression by encoding the bin index of each context using the corresponding context population in (5). For the above scheme, we may need to spend extra bits to represent the size of each context population.

We propose two quantization schemes for better compression in which we first prune the context tree using (5). In our first scheme, we use roughly $1.772(N/|\hat{S}|)^{0.5}$ quantization bins for encoding each bin index. With this scheme, there is no need to encode context populations, because the quantizer will perform well on average. In our second scheme, we use a 2-level quantization scheme in which all contexts whose population is below some threshold τ are processed with a small coarse quantizer whose index can be encoded with a minimal number of bits, while contexts whose population exceeds τ are processed with a larger fine quantizer. Both of our quantization schemes have only a minor impact on the run time over the original quantization scheme (5). Note that Theorems 1 and 2 hold for the modifications we propose for the two new quantizers.

Using a reduced-length quantizer might encourage the MDL optimization to increase the number of states, which could further improve the compression. Indeed, we tried to rerun the MDL optimization procedure with $N/|\hat{S}|$ instead of N in (5), but this increased the run time by 30 – 40% while yielding modest compression gains. Therefore, we do not recommend rerunning the MDL optimization.

Vector quantization [12, 24], which can be used when the binary symbols are clustered to form a larger alphabet, has the potential to improve the compression performance by up to an additive constant of $O(\frac{K}{2} \log(K))$ [25, 26] over the scalar quantization based compression algorithm. However, in order to maintain $O(N/B)$ time complexity, we may need to reduce the maximum depth D .

IV. NUMERICAL RESULTS

Having described the PTP-MDL algorithm, we have set the stage to describe our numerical results with a prototype implementation [2]. After surveying our simulation setting, we compare the compression ratio and the throughput of the PTP-MDL algorithm with two types of algorithms: (i) high quality compressors and (ii) fast

Compression algorithm	Compression ratio (bits/byte)			Average throughput (Mbps)
	E.Coli	bible.txt	world 192.txt	
LZ77a (32KB)	2.35	2.32	2.32	31
LZ77b (4MB)	2.27	1.93	1.72	36
BWT	2.16	1.67	1.58	36
NanoZip	1.97	1.42	1.25	84
Gipfeli	6.00	7.49	7.27	826
LZ4	5.45	4.14	3.98	1515
Snappy	3.73	3.93	4.02	2025
PTP-MDL (B=1)	1.98	2.15	2.45	2
“- (B=10)	1.99	2.36	2.85	17
“- (B=100)	1.99	2.57	3.20	138
“- (B=1000)	2.01	3.09	3.77	908

Table I: Performance comparison for different compressors.

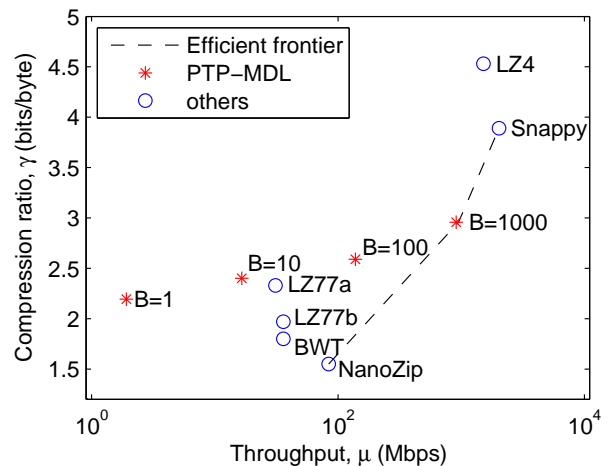


Figure 4: Compression ratio vs. throughput for different compressors.

compression algorithms. The PTP-MDL algorithm offers competitive performance with both types of algorithms. We show the trade-off between compression ratio and throughput of the PTP-MDL algorithm as a function of B . Finally, we show the improvement in compression performance of the PTP-MDL algorithm using the quantization schemes proposed in Section III-F.

A. Simulation setting

We have developed a serial (non-parallel) implementation in C++, which serves as a prototype that allows us to evaluate anticipated performance of a GPU implementation, which is ongoing work. The algorithm treats any data as a binary bit stream; in future work, we plan to apply techniques that exploit the byte nature of real-world data [27]. Although the parallel parts of the algorithm run sequentially in the current implementation,

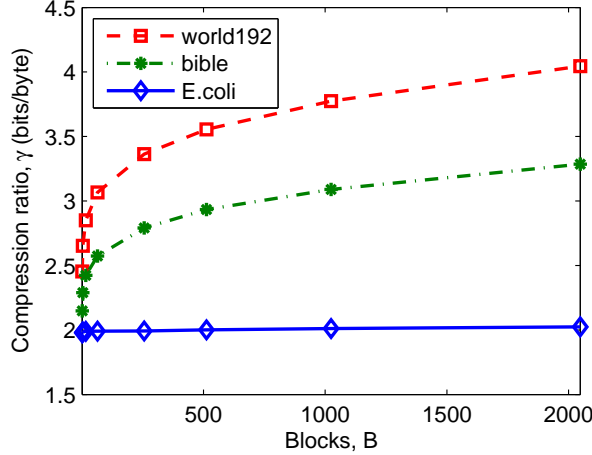


Figure 5: Compression ratio vs. number of blocks.

we give a predicted time performance as

$$t_{\text{est}}(B) = t_s + \frac{t_{\text{sp}}}{\eta B}, \quad (13)$$

where $t_{\text{est}}(B)$ is the estimated time for executing the algorithm using B PUs, t_s is the time required to execute the serial part of the algorithm, t_{sp} is the sequential time required for executing the parallel part of the algorithm, and $\eta \in [0, 1]$ is the *efficiency of parallelization*.

The *compression ratio*, γ , is defined as the average number of output bits in the compressed data required to represent one input byte of uncompressed data. Note that the lower the γ , the better the compression. The *throughput*, $\mu = \frac{N}{t_{\text{est}}(B)}$, is measured in megabits per second (Mbps); the higher the throughput, the shorter the run time. We assume $\eta = 0.2$ in our simulations to provide a conservative estimate of the throughput unless specified otherwise.

B. PTP-MDL vs. other algorithms

We compare the compression ratio and throughput of several algorithms for files from the large Canterbury corpus (<http://corpus.canterbury.ac.nz/descriptions>) in Table I and Figure 4. The best trade-off is achieved for algorithms that have the lowest compression ratio and the highest throughput, and are represented by points that are closest to the bottom right corner in Figure 4. We ran the PTP-MDL algorithm for $B \in \{1, 10, 100, 1000\}$. Although running $B = 1000$ parallel units may seem ambitious, GPUs with thousands of cores are already commonplace. PTP-MDL is compared with high quality compressors such as Lempel-Ziv coding [12] with a 32KB dictionary size (LZ77a), Lempel-Ziv coding with a 4MB dictionary size (LZ77b), the Burrows-Wheeler

transform (BWT) [28], and the context-based NanoZip algorithm (<http://nanozip.net/>), as well as fast algorithms such as LZ4 [29], Gzip [4], and Snappy [5]. For the E.coli file, PTP-MDL and NanoZip have the lowest compression ratio. Compression with NanoZip also provides the lowest compression ratios for the two text files, bible.txt and world192.txt.

The PTP-MDL algorithm has a competitive compression ratio although this algorithm is implemented for binary symbols, whereas the other algorithms are designed for 8 bit symbols, to the best of our knowledge. LZ4, Gzip, and Snappy, which are speed optimized approximations of LZ77 [12], did not do well in compression ratio performance. In addition to a competitive compression ratio, as B increases, the throughput of the PTP-MDL algorithm is comparable to the throughputs of fast data compression algorithms such as LZ4 (≈ 1515 Mbps) and Snappy (≈ 2025 Mbps) (Table I), which can be used in big data problems.

We highlight that the efficient frontier in Figure 4 reflects the optimal trade-off between compression ratio and throughput. PTP-MDL extends the efficient frontier for large B .

C. Compression ratio and throughput vs. B

PTP-MDL γ performance vs. B: Figure 5 illustrates the impact of the number of blocks B on the compression ratio γ . For a simple source such as E.coli, where D is small, the compression ratio increase with B is modest. However, there is a non-linear behavior for more complicated data such as English text. For bible.txt and world192.txt, the compression ratio γ deteriorates rapidly for small B . This deterioration could be due to the decrease in maximum depth available for the tree source given by $O(\log(N/B))$, which may impact the coding length as B increases.

PTP-MDL μ performance vs. B: Figure 6 illustrates the impact of the number of blocks B on the average compression throughput μ for files from the large Canterbury corpus for two efficiencies of parallelization, $\eta_1 = 0.5$ and $\eta_2 = 0.2$. For small B , the speed up is linear, and as B increases, the speed up slows down. This trend is due to *Amdahl's law* [30] given in (13). When B is low, the fraction of serial execution time is insignificant compared to parallel execution time. However, when B is large, the parallel fraction of execution time is reduced, which introduces nonlinearity in the throughput μ for large values of B . The result for decompression throughput is similar to that of compression throughput, and thus omitted for brevity.

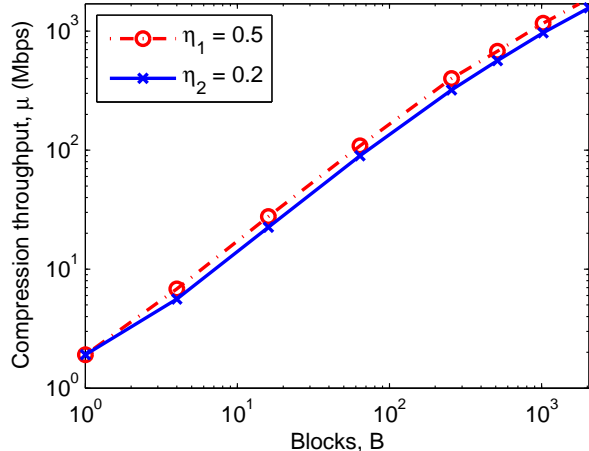


Figure 6: Compression throughput vs. number of blocks.

D. Compression ratio for 2 level quantization scheme

Table II shows percentage improvements in the compression ratio γ obtained using the two quantization schemes discussed in Section III-F w.r.t. the original quantization scheme (5) for files from the large Canterbury corpus. Among the two quantization schemes, the second scheme based on 2-level quantization narrowly outperforms the first scheme. From Table II, it can be seen that the E.coli file, which PTP-MDL already compresses almost as well as the state of the art NanoZip (Table I), does not improve much. For the text files bible.txt and world192.txt, the 2-level quantization scheme typically improves compression by 1–2%. Note, however, that the compression improvement is modest for large B .

V. CONCLUSION

In this paper, we have proposed a parallel two-pass minimum description length (PTP-MDL) algorithm for compressing context tree sources of depth $D = \log(N/B)$. The algorithm can compress and decompress in $O(N/B)$ time while achieving a redundancy within $B \log(N/B)$ bits of Rissanen’s lower bound on universal compression performance [19]. We further improved the PTP-MDL algorithm using a 2 level quantization scheme. In future work, we plan to explore and analyze the use of multi-level quantizers to improve the compression performance without degrading the time complexity.

Our numerical results show that the compression ratio γ of our algorithm for real-world data is comparable to existing universal data compressors. Moreover, the throughput of PTP-MDL scales well with the number of parallel units, even for large B . We speculate that

B	Quantization scheme 1			Quantization scheme 2		
	E .Coli	bible .txt	world 192.txt	E .Coli	bible .txt	world 192.txt
1	0.03%	1.79%	4.37%	0.01%	1.91%	4.72%
10	0.01%	1.09%	2.28%	0.01%	1.14%	2.50%
100	0.01%	0.58%	1.05%	0.01%	0.64%	1.35%
1000	0.00%	0.16%	0.37%	0.00%	0.21%	0.44%

Table II: Percentage improvements in the compression ratio γ obtained using quantization schemes in Section III-F w.r.t. the original quantization scheme (5).

exploiting the byte nature of real-world data can further improve the compression ratio γ .

The PTP-MDL algorithm has the useful property of random access [16], where any part of the compressed file can be decompressed without decompressing the entire file. This property is useful in applications where the compressed file is large, and only part of the file is needed to service a query. The JPEG2000 compression standard uses the random access property of the Embedded Block Coding with Optimized Truncation (EBCOT) based lossless encoder to selectively reconstruct the region of interest [31]. The PTP-MDL algorithm can replace the lossless encoder in JPEG2000 to improve the throughput without losing the random access property. It might be possible to also apply PTP-MDL to lossless image compression using the two-dimensional contexts of Weinberger et al. [32], but in the image setting the number of contexts is often prohibitively large, and it is not clear whether comprehensive changes to PTP-MDL would be needed.

Finally, although the two-pass approach may seem costly in applications where data is streamed (online compression), we can design the compression system that minimizes the impact of redundancy by dividing the data into reasonably large blocks such that the entire block can be processed simultaneously in random access memory (RAM). With currently available hardware circa 2015, our proposed algorithm has the potential to compress data of the order of gigabytes (GBs) simultaneously. Another related direction for future inquiry is to lower memory utilization in the parallel compression design space [9].

ACKNOWLEDGEMENTS

We thank Yoram Bresler and Mehmet Kıvanç Mıhçak for numerous discussions relating to this work; Frans Willems for the arithmetic code implementation; and the reviewers, the editor, Yanting Ma, Jin Tan, and Junan Zhu for their careful evaluation of the manuscript.

REFERENCES

- [1] N. Krishnan, D. Baron, and M. K. Mihçak, "A parallel two-pass MDL context tree algorithm for universal source coding," in *Proc. Int. Symp. Inf. Theory (ISIT)*, July 2014.
- [2] N. Krishnan and D. Baron, "Performance of parallel two-pass MDL context tree algorithm," in *Proc. IEEE Global Conf. Signal Inf. Process.*, Atlanta, GA, Dec. 2014.
- [3] S. Arming, R. Fenhkhuber, and T. Handl, "Data compression in hardware – the Burrows-Wheeler approach," in *IEEE Int. Symp. Des. Diagnostics Electron. Circuits Syst.*, Apr. 2010, pp. 60–65.
- [4] R. Lenhardt and J. Alakuijala, "Gipfeli - high speed compression algorithm," in *Proc. Data Compression Conference (DCC)*, Apr. 2012, pp. 109–118.
- [5] S. Gunderson, "Snappy, A fast compressor/decompressor," code.google.com/p/snappy/.
- [6] L. M. Reinhold, "QuickLZ website," <http://www.quicklz.com/>.
- [7] A. Hidayat, "FastLZ website," <http://fastlz.org/>.
- [8] P. Franaszek, J. Robinson, and J. Thomas, "Parallel compression with cooperative dictionary construction," in *Proc. Data Compression Conf. (DCC)*, Mar. 1996.
- [9] A. Beirami and F. Fekri, "On lossless universal compression of distributed identical sources," in *Proc. Int. Symp. Inf. Theory (ISIT)*, July 2012, pp. 561–565.
- [10] M. L. A. Stassen and T. J. Tjalkens, "A parallel implementation of the CTW compression algorithm," in *Proc. 22d Benelux Symp. Inf. Comm.*, May 2001, pp. 85–92.
- [11] F. M. J. Willems, Y. M. Shtarkov, and T. J. Tjalkens, "The context tree weighting method: Basic properties," *IEEE Trans. Inf. Theory*, vol. 41, no. 3, pp. 653–664, May 1995.
- [12] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, New York, NY, USA: Wiley-Interscience, 2006.
- [13] F. M. J. Willems, "Some challenges in source coding," in *Proc. 3rd ITG Conf. Source Channel Coding*, Jan. 2000, pp. 245–249.
- [14] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, no. 5, pp. 465–471, Sept. 1978.
- [15] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, The MIT Press, Cambridge, MA, 2009.
- [16] S. Kreft and G. Navarro, "LZ77-like compression with fast random access," *Data Compression Conference (DCC)*, 2010, pp. 239–248, Mar. 2010.
- [17] D. Baron, "Fast parallel algorithms for universal lossless source coding," Feb. 2003, Ph.D. thesis, UIUC.
- [18] M. J. Weinberger, N. Merhav, and M. Feder, "Optimal sequential probability assignment for individual sequences," *IEEE Trans. Inf. Theory*, vol. 40, no. 2, pp. 384–396, Mar. 1994.
- [19] J. Rissanen, "Fisher information and stochastic complexity," *IEEE Trans. Inf. Theory*, vol. 42, no. 1, pp. 40–47, Jan. 1996.
- [20] D. Baron and Y. Bresler, "An O(N) semipredictive universal encoder via the BWT," *IEEE Trans. Inf. Theory*, vol. 50, no. 5, pp. 928–937, May 2004.
- [21] P. A. J. Volf and F. M. J. Willems, "A study of the context tree maximizing method," in *Proc. 16th Benelux Symp. Inf. Theory, Nieuwerkerk IJssel, Netherlands*, May 1995, pp. 3–9.
- [22] F. M. J. Willems, Y. M. Shtarkov, and T. J. Tjalkens, "Context-tree maximizing," in *Proc. Conf. Inf. Sci. Syst.*, Mar. 2000, pp. 7–12.
- [23] D. Baron, Y. Bresler, and M. K. Mihçak, "Two-part codes with low worst-case redundancies for distributed compression of Bernoulli sequences," in *Proc. Conf. Inf. Sciences Systems*, Mar. 2003.
- [24] A. Gersho and R. M. Gray, *Vector quantization and signal compression*, Kluwer, 1993.
- [25] Q. Xie and A. R. Barron, "Minimax redundancy for the class of memoryless sources," *IEEE Trans. Inf. Theory*, vol. 43, no. 2, pp. 646–657, Mar. 1997.
- [26] A. Beirami and F. Fekri, "Results on the redundancy of universal compression for finite-length sequences," in *Proc. Int. Symp. Inf. Theory (ISIT)*, July 2011, pp. 1504–1508.
- [27] F. M. J. Willems, "The context-tree weighting method: Extensions," *IEEE Trans. Inf. Theory*, vol. 44, no. 2, pp. 792–798, Mar. 1998.
- [28] M. Burrows and D. J. Wheeler, *A block-sorting lossless data compression algorithm*, 1994.
- [29] "LZ4, Extremely Fast Compression algorithm," code.google.com/p/lz4/.
- [30] Y. Solihin, *Fundamentals of Parallel Computer Architecture*, Solihin Publishing and Consulting LLC, 2009.
- [31] D. Taubman, "High performance scalable image compression with EBCOT," *IEEE Trans. Image Process.*, vol. 9, no. 7, pp. 1158–1170, Jul. 2000.
- [32] M. J. Weinberger, J. J. Rissanen, and R. B. Arps, "Applications of universal context modeling to lossless compression of gray-scale images," *IEEE Trans. Image Process.*, vol. 5, no. 4, pp. 575–586, Apr. 1996.



Nikhil Krishnan (S'13) was born in Trivandrum, India, in 1984. He received the B.Tech. degree in electronics and communication engineering from the College of Engineering Trivandrum, India, in 2006, and the M.Tech. degree in electrical engineering from the Indian Institute of Technology (IIT) Bombay, India, in 2010.

From 2006 to 2008, he worked as a Scientist at the Indian Space Research Organization, India, in navigation systems. From 2010 to 2012, he worked at Samsung India Software Operations in image processing. Since 2012, he has been pursuing his Ph.D. in the Electrical and Computer Engineering Department at NC State University in Raleigh. His current research interests include information theory, signal processing, compression, and performance of algorithms.



Dror Baron (S'99-M'03-SM'10) received the B.Sc. (summa cum laude) and M.Sc. degrees from the Technion - Israel Institute of Technology, Haifa, Israel, in 1997 and 1999, and the Ph.D. degree from the University of Illinois at Urbana-Champaign in 2003, all in electrical engineering. From 1997 to 1999, Dr. Baron worked at Witcom Ltd. in modem design. From 1999 to 2003, he was a research assistant at the University of Illinois at Urbana-Champaign, where he was also a

Visiting Assistant Professor in 2003. From 2003 to 2006, he was a Postdoctoral Research Associate in the Department of Electrical and Computer Engineering at Rice University, Houston, TX. From 2007 to 2008, he was a quantitative financial analyst with Menta Capital, San Francisco, CA, and from 2008 to 2010 he was a Visiting Scientist in the Department of Electrical Engineering at the Technion - Israel Institute of Technology, Haifa. Since 2010, Dr. Baron has been an Assistant Professor in the Electrical and Computer Engineering Department at NC State University. Dr. Baron's research interests combine information theory, signal processing, and fast algorithms; in recent years, he has focused on compressed sensing.

Dr. Baron was a recipient of the 2002 M. E. Van Valkenburg Graduate Research Award, and received honorable mention at the Robert Bohrer Memorial Student Workshop in April 2002, both at the University of Illinois. He also participated from 1994 to 1997 in the Program for Outstanding Students, comprising the top 0.5% of undergraduates at the Technion.